테스트 환경 개선을 위한 테스트케이스 자동 생성 알고리즘 연구*

원예인**, 이장행**, 김지운**, 권오승***, 손윤식**
**동국대학교 컴퓨터공학과
***(주) 스패로우 R&D 센터 경영학박사

A Study on the Testcase Automatic Generation Algorithm for Improving Test Environment

Yein Wone**, Janghaeng Lee**, Jiwoon Kim**, Ohseung, Kwon***, Yunsik Son**
**Dept. of Computer Science&Engineering, Dongguk University
***R&D Center, Sparrow Co.

요 약

소프트웨어 개발 주기의 마지막 단계인 테스팅 단계의 가치 및 중요도가 높아짐에 따라, 소프트웨어 테스팅 자동화 도구에 대한 수요 역시 비례하여 증가하는 추세이다. 이에 테스트 자동화 도구를 개발하여 테스트 기간 및 소요 인력, 비용을 감소시킴과 동시에 테스트 성능은 유지 혹은 개선시켜 소프트웨어 품질을 향상시키고자 한다. 본 연구에서는 Rule-based random testing 을 위한 테스트케이스 자동 생성 프로그램을 개발하고 이를 위한 테스트케이스 자동 생성 알고리즘을 제안한다.

1. 서론

소프트웨어 기능 테스트는 개발자의 노력과 시간이 많이 소요되며 작업 특성상 지루하고 반복적인 것이 대부분이다[1]. 이에 테스트케이스를 자동으로 생성하는 기능을 가진, 신뢰할 수 있는 테스트케이스 생성 자동화 도구를 개발하여 수동적인 업무를 줄이고, 시간 및 비용을 감소시키는 효과를 얻는다[2]. 이는 소프트웨어의 신뢰성을 높이고 테스팅 시간 일부를 개발하는 데 할애할 수 있어 소프트웨어 품질 향상에 도움을 준다[3]. 테스트 케이스를 생성하는 방법은 정적 분석, 조합 및 랜덤 생성 등 다양한 방식이 있다[4]. 이중에 최근 brute-force 테스트의 한 측으로 사용되는 랜덤 생성 알고리즘에 대해서 연구할 필요성이 대두되었다[5].

본 연구는 사용자가 원하는 보다 유의미한 테스트케이스를 생성하는 알고리즘을 개발하는데 목표로 한다. 테스트 케이스를 자동으로 생성하는 랜덤 알고리즘은 랜덤 규칙을 이용한다. 랜덤 규칙 기반의 랜덤 생성 방식은 테스트케이스를 무조건적 난수를 생성하는 것이 아닌 테스트의 효율을 높일 수 있는 의미 있는 조건적 난수를 생성하는데 의미를 둔다.

*본 연구는 과학기술정보통신부 및 정보통신기획평가원의 SW 중심대학지원사업의 연구결과로 수행되었음(2016-0-00017)

2. 기존 테스트케이스 자동 생성 연구

테스트케이스 자동 생성에 관한 연구에는 문맥 자유 문법인 CFG 로 테스트 케이스 자동 생성 연구[6]가 있다. 해당 연구는 CFG 에 의해 자동 생성되는 테스트 케이스의 집합의 크기를 통제, 테스트의 주요 범위를 커버하기 위하여 문맥 자유 문법을 보강하는 문법외적 요소인 태그들을 개발, 실험한다. 테스트케이스를 자동으로 생성하는 방법에는 이 외에도 combination based 생성, random based 생성, pairwise based 생성 등 다양하다. 본 논문에서는 자동 생성 기법 중에서 랜덤 기반으로 테스트케이스를 자동 생성하는 알고리즘을 연구하고 제안한다. 제안한 알고리즘으로 rule-based random testing 을 위한 테스트케이스 자동 생성 프로그램을 개발하고 테스트 환경 개선을 목적으로 한다.

3. 본론

3.1. 필드 유형 분석

랜덤 테스팅을 위한 테스트케이스 생성 알고리즘을 도출하기 위해 먼저 업무 화면에서 사용되는 필드 유형을 분석하였다. 분석 대상 화면은 동국대학교 학적관리 시스템 화면, Fasoo 기업 인사관리 화면과 법원의 사건 번호 등록 화면으로 한정하였다.

(그림 1) 동국대학교 학적 관리 시스템 화면

(그림 1)을 포함한 총 3 개의 화면에서 총 110 개의 필드 중에서 중복 필드 96 개를 제거하여 24 개의 필드 유형을 추출하였고 이 들을 분석하여 7 개의 랜덤 알고리즘을 도출하였다. 중복 필드 제거는 다음 규칙을 적용하였다.

규칙 1. 의미 중복 필드 제거
첫 번째로, 의미가 중복되는 필드를 제거하였다. 예로, 날짜, 생년월일과 같이 동일한 의미를 가지는 필드는 하나로 통합하였다.

규칙 2. 형식 중복 필드 제거
두 번째로, 형식이 같은 필드를 제거하였다. 예로, 남녀 구분, 예/아니오, 내국인/외국인 등은 의미는 같지 않지만 두 개 선택지 중에 하나를 선택하는 형식이 같다. 다음과 같이 형식이 중복되는 필드를 하나로 통합하였다.

이와 같이 규칙 1 과 규칙 2 를 적용한 결과 총 24 개의 필드 유형이 도출되었다.

<표 1> 분석 대상 화면에서 추출한 필드 유형 목록

순번	필드 유형	순번	필드 유형
1	숫자	13	우편 번호
2	스트링	14	이름
3	구분자 지정 스트링	15	이메일
4	범위 지정 인티저	16	일련번호
5	N 자리 스트링	17	주민등록번호
6	계좌번호	18	주소
7	구분자 스트링	19	지역 전화번호
8	날짜	20	체크박스
9	돈	21	학교
10	콤보박스	22	휴대폰 번호
11	라디오버튼	23	N 자리 숫자
12	사건번호	24	Y/N

3.2. 랜덤 알고리즘 제안

랜덤 생성 알고리즘은 24 개의 필드 유형에 대해 테스트케이스 데이터를 모두 생성할 수 있어야 한다는 요건을 정하였고 이를 만족하도록 아래 규칙을 적용하여 24 개의 필드를 커버할 수 있는 랜덤 알고리즘 유형 7 개를 도출하였다.

규칙 3. 데이터 도메인을 숫자와 문자로 1 차 구분
규칙 4. 숫자 도메인을 정수 도메인과 실수 도메인으로 구분
규칙 5. 문자 도메인을 형식 유무로 구분
규칙 6. 형식 존재 문자 도메인을 특정 조건 유무로 구분

위의 규칙 3 부터 규칙 6 까지 적용한 결과 7 개의 알고리즘 유형으로 정리되었고, 표 2 와 같다.

<표 2> 랜덤 알고리즘 유형 목록

유형 이름
RandomInt
RandomDouble
RandomString
RandomDate
TypeDefinedString
SpecificString
DBReferencedNumber

도출한 랜덤 알고리즘 유형이 <표 1> 에 나와있는 목록을 모두 커버할 수 있는지 정리하면 <표 3> 과 같다.

<표 3> 랜덤 알고리즘 유형과 필드 유형 매칭

랜덤 알고리즘 명	해당 필드 유형
RandomInt	숫자, 범위 지정 인티저, 드롭다운, 라디오버튼, 체크박스, N 자리 숫자, Y/N
RandomDouble	-
RandomString	스트링, N 자리 스트링, 우편번호, 일련번호
RandomDate	날짜
TypeDefinedString	구분자 지정 스트링, 계좌번호, 구분자 스트링, 돈
SpecificString	사건번호, 주민등록번호, 주소, 지역전화번호, 학교, 이메일, 휴대폰번호
DBReferencedNumber	이름

3.3 필드 커버 예시

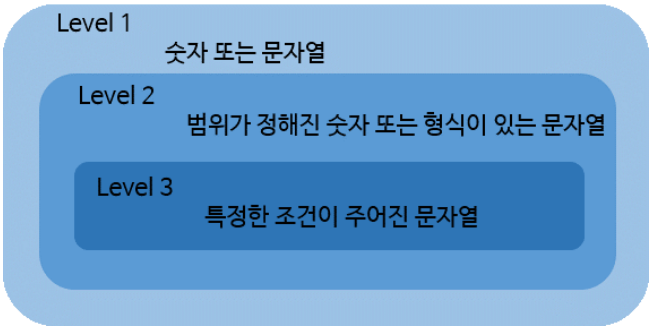
이후 설명에서 알고리즘 유형 중 TypeDefiendString 으로 테스트 대상 화면의 주민등록번호 필드, IP 주소 필드, 전화번호 필드를 커버하는 예시를 들었다.

주민등록번호, IP 주소, 전화번호의 공통점은 숫자와 특수문자로 이루어졌다는 점이다. 하지만 차이점은 숫자의 개수, 숫자의 길이, 구분자의 개수가 다르다는 점이다. 주민등록번호는 6 자리 숫자. '- ' 특수문자, 7 자리 숫자로 이루어지고 IP 주소는 숫자로 이루어진 4 개의 영역을 '.' 특수문자로 구별한 문자열이다. 전화번호는 3 개의 영역을 가지며 첫번째 영역은 지역번호를 의미하고 각 영역은 보통 '- '특수문자로 구별된다. 이 세 필드를 하나의 알고리즘으로 커버하기 위해 구별되는 점을 알고리즘의 인풋 파라미터로 받는다. 따라서 TypeDefinedString 의 인풋 파라미터로 최대 길이, 문자열의 형식(숫자, 영어), 구분자, 영역의 수, 각 영역의 길이를 설정한다.

위 예시와 같은 방법으로 <표 3>의 각각의 알고리즘으로 매칭된 필드 유형들을 커버할 수 있다. 이는 7 개의 알고리즘이 총 24 개의 필드유형을 커버할 수 있고 테스팅 대상 화면에 있는 110 개의 필드를 커버할 수 있음을 의미한다.

3.4 생성되는 데이터 레벨 분류

개발자는 생성되는 테스트케이스 데이터들의 구체화 수준을 설정하여 테스트케이스를 생성할 수 있다. 테스트 데이터들은 3 개의 레벨로 분류하였다.



(그림 2) 테스트케이스 데이터의 구체화 레벨

자동으로 생성되는 테스트 데이터들은 (그림 2)에서와 같이 3 개의 레벨로 분류할 수 있다. 레벨이 높아질수록 구체화 수준이 높아져 생성되는 데이터가 실제와 유사하다. 집합의 개념으로 보면 레벨 1 은 레벨 2 의 데이터를 모두 포함하고, 레벨 2 는 레벨 3 을 데이터를 모두 포함한다. 레벨 1 은 숫자와 문자열로 데이터를 구분하는 것, 레벨 2 는 범위가 정해진 숫자와 형식이 있는 문자열로 데이터를 구분하는 것, 레벨 3 은 특정한 조건이 주어진 문자열로 데이터를 구분하는 것이다.

예를 들어, 주민등록번호 필드가 있을 때 RandomString 알고리즘으로 레벨 1 수준에서 테스트케이스를 생성하면 숫자와 특수문자로 이루어진 234`034-0*3^%32342 라는 데이터가 생성된다. 이는 주민등록번호가 숫자와 특수문자로 이루어져 있다는 정보를 기반으로 생성된 데이터이다. TypeDefinedString 알고리즘으로 좀더 구체화된 레벨 2 수준에서 테스트케이스를 생성하면 483944-8463882 와 같은 테스트케이스가 생성된다. 이 데이터는 주민등록번호가 6 자리 숫자, '- '특수문자, 7 자리 숫자로 이루어져 있다는 정보로 생성되어 레벨 1 에서의 데이터보다 좀더 주민등록번호에 가깝다. 하지만 앞에 영역의 6 자리 숫자는 생년월일 형식에 벗어나고 두번째 영역의 첫번째 숫자는 성별 구별 숫자(1,2,3,4)로 이루어지지 않은 데이터이다. SpecificString 알고리즘으로 레벨 3 수준에서 데이터를 생성하면 990616-1748574 와 같은 형식의 데이터가 생성되는데 이는 실제 주민등록번호 형식을 갖고 생성된 가장 주민등록번호에 가까운 데이터이다. 임의의 한글 문자열을 생성하는 경우에 상위 레벨일수록 존재하는 글자들로 구성된다. 레벨 1,2 수준에서 한글 문자열 생성은 한글 유니코드상에서 랜덤의 문자를 뽑아 조합하는 방식으로 이루어진다. 이 수준에서는 '궸쉦걊' 과 같은 테스트케이스 자체의 가치는 있지만 실제 사용되지 않는 단어로, 테스팅하기에는 적절하지 않다고 판단될 가능성이 높은 문자열이 생성된다. 레벨 3 수준에서 더 의미 있는 문자열을 생성하기위해 lorem ipsum 생성 기법을 사용하였다. lorem ipsum 생성 기법을 사용하여 '다하여 바다' 와 같은 실제 존재하는 단어들로 조합된 문자열을 생성한다.

이와 같이 생성하려는 테스트 데이터의 구체화 수준에 따라 레벨을 정하여, 데이터 생성을 위하여 규칙을 설정하는 시간과 생성되는 데이터의 품질 간 trade-off 를 테스터가 선택할 수 있도록 하였다.

3.5 제안한 알고리즘의 평가 및 분석

본 논문에서 제안한 7 개의 알고리즘으로 110 개의 필드를 커버하는 것은 가능하다. 하지만 7 개의 알고리즘으로 커버하지 못하는 필드가 존재할 수도 있다는 한계가 있다. 이에 대한 방안으로는 커버 불가능한 필드에 대해서는 레벨을 낮춰 데이터를 생성하거나 2.2 의 방식을 통하여 새로운 알고리즘을 생성한다.

4. 결 론

본 연구에서는 사용자가 입력한 랜덤 생성 규칙을 기반으로 랜덤 테스팅을 위한 테스트 케이스를 자동으로 생성하는 소프트웨어 테스트를 위한 테스트케이스 자동 생성 알고리즘을 개발하였다. 이를 통하여 소프트웨어 개발자들의 편의성을 보다 증대시키고 소프트웨어 품질을 향상시켜 소비자들의 만족도 증가를 기대할 수 있을 것이다. 연구에서 개발한 알고리즘은 소프트웨어 테스트 도구에 포함될 예정이다. 본 연구

에서 제안한 알고리즘의 한계점으로 지목된 커버 불가능한 영역의 해결에 대해서는 향후 연구 과제로 두어 계속하여 연구를 진행할 예정이다.

참고문헌

[1] C. Kaner, J. L. Falk, and H. Q. Nguyen, Testing Computer Software, 2nd ed. New York: Van Nostrand Reinhold, 1993.
[2] Jungsup Oh, Jongmoon Baik and Sung-Hwa Lim, A Model Independent S/W Framework for Search-Based Software Testing
[3] R. D. Craig and S. P. Jaskiel, Systematic Software Testing. Boston: Artech House, 2002.
[4] L. Copeland, A Practitioner's Guide to Software Test Design. Boston, MA: Artech House Publishers, 2003.
[5] C. Kaner, J. Bach, and B. Pettichord, Lessons Learned in Software Testing: A Context Driven Approach. New York: John Wiley & Sons, Inc., 2002.
[6] 윤성희. (2009). CFG 를 이용한 소프트웨어 테스트 케이스의 자동 생성. 한국산학기술학회논문지, Vol. 10, No. 5, pp. 985-991